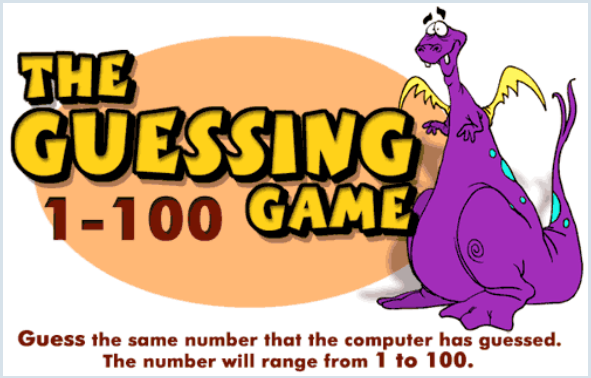
Condições e Decisões

Fernanda Barbosa
Fernando Birra
Luís Caires
Armanda Rodrigues

Programas com Decisões

- Neste capítulo, vamos estudar como definir em Java programas que tomam decisões e executam acções em alternativa como resultado de verificar condições.



- No caminho, serão introduzidas noções importantes:
  - A instrução condicional: **`if-then-else`**
  - O bloco de instruções (ou instrução composta)
  - O programa principal simples
  - A operação de output **`System.out.println`**

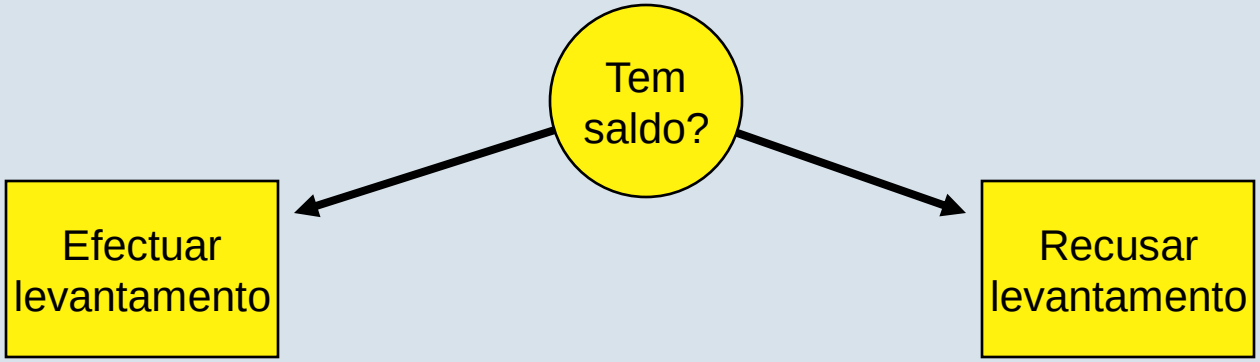
Recorde a Conta Bancária

- Uma conta bancária é:
  - Um “depósito” de dinheiro
  - A quantidade de dinheiro na conta chama-se saldo
  - O saldo pode ser positivo (credor ou nulo) ou negativo (devedor)
- Quando a conta é criada
  - Se não indicarmos nada, o seu saldo é zero
  - Em alternativa, podemos indicar um valor inicial para o saldo
- Operações reconhecidas (interface de BankAccount):

```
public void deposit(int amount)
public void withdraw(int amount)
public int getBalance()
public boolean redZone()
```

Cenário mais realista

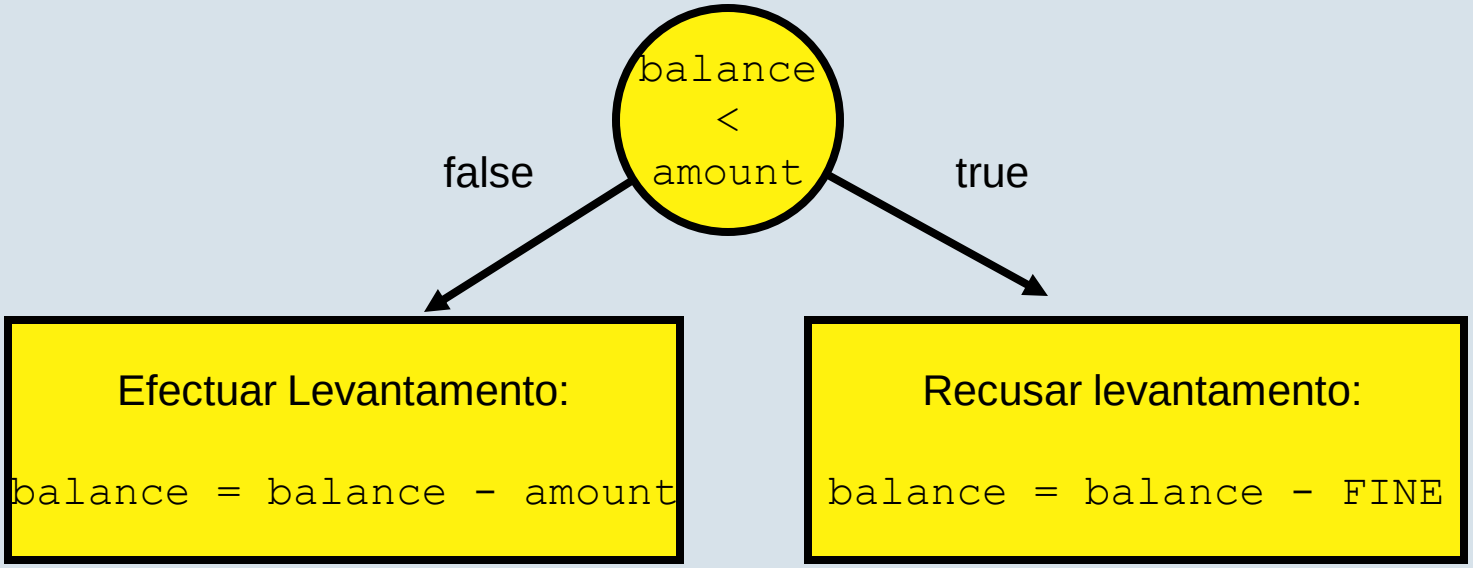
- Sempre que um cliente pretende levantar dinheiro de uma conta bancária, o banco pode tomar uma de duas decisões alternativas
  - Aceitar o levantamento
    - Porque a conta **tem** saldo suficiente
  - Recusar ou modificar o levantamento
    - Porque a conta **não tem** provisão suficiente



Procedimento Bancário

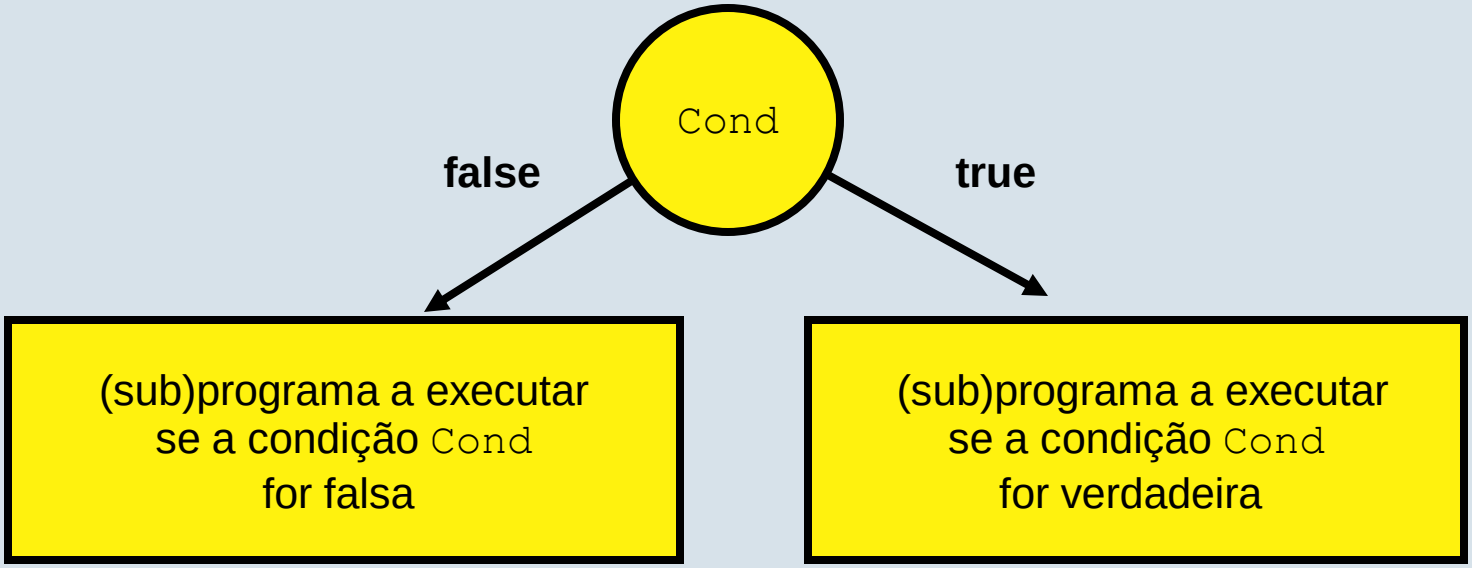
- Vamos considerar que o banco segue as seguintes regras de negócio:
  - O levantamento só efectuado caso o valor a levantar seja inferior ou igual ao valor do saldo
  - Qualquer tentativa de levantamento de um valor sem provisão na conta leva à aplicação de uma multa
- O aspecto mais importante a reter aqui é que processar um levantamento requer uma **decisão** entre duas acções **alternativas e exclusivas**.

Procedimento Bancário



- O levantamento pedido (`amount`) só é efectuado caso o valor a levantar seja inferior ou igual ao valor do saldo
- Qualquer tentativa de levantamento de um valor sem provisão na conta leva à aplicação de uma multa (`FINE`)

Decisões em Programação



- Uma decisão envolve a avaliação de uma expressão booleana `Cond`, uma condição que produz `true` ou `false`
- Qualquer decisão envolve determinar o que deve o programa fazer em qualquer uma das situações.
- Para o programa poder decidir em qualquer circunstância, o programador tem que prever as duas possibilidades

A instrução `if-else`

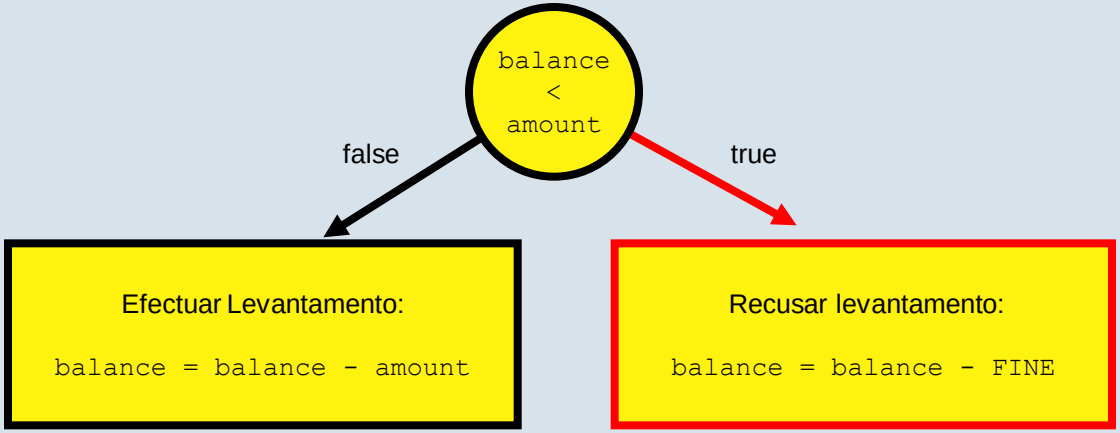
- Na linguagem Java, as decisões exprimem-se usando a instrução **`if-else`**

```
if ( condition )
    parteTHEN;
else
    parteELSE;
```

- A expressão ***`condition`*** tem que ser uma expressão booleana: uma condição que produz **`true`** ou **`false`**
- Como é executada a instrução **`if-else`** ?
  - Primeiro, a condição ***`condition`*** é avaliada e produz um valor.
  - Se o valor é **`true`**, então o programa executa apenas a ***`parteTHEN`***
  - Se o valor é **`false`**, então o programa executa apenas a ***`parteELSE`***

A instrução `if-else`

```
if (balance < amount )
   balance = balance – FINE;
       else
   balance = balance - amount;
```




```
if (condicao)
        parteTHEN;
   else
        parteELSE;
```

A instrução `if-else`

```
if (balance < amount )
   balance = balance - FINE;
       else
   balance = balance - amount;
```

Se *condicao* for verdadeira,

A *parteTHEN* é executada,

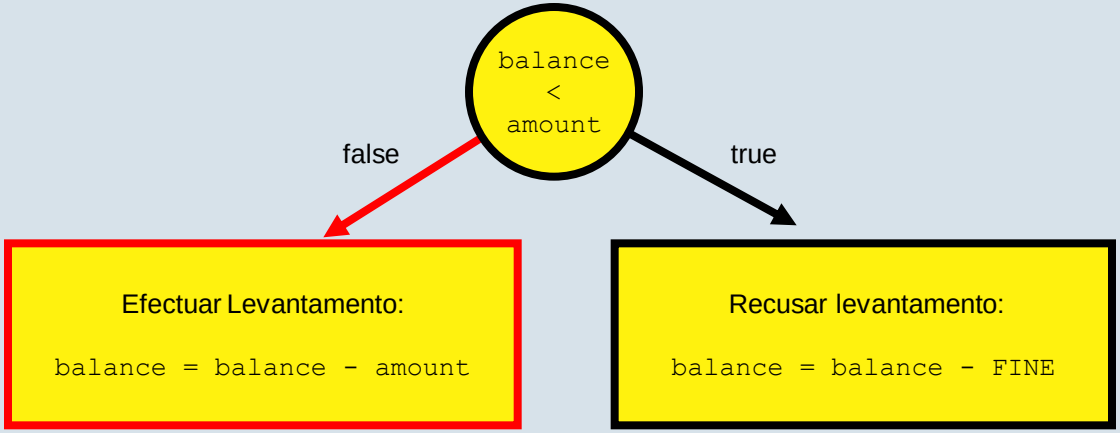
caso contrário,

é executada a *parteELSE*.

```
if (condicao)
        parteTHEN;
   else
        parteELSE;
```

A instrução `if-else`

```
if (balance < amount )
   balance = balance - FINE;
      else
   balance = balance - amount;
```




```
if (condicao)
        parteTHEN;
   else
        parteELSE;
```

A instrução `if-else`

```
if (balance < amount )
   balance = balance - FINE;
       else
   balance = balance - amount;
```

Se *condicao* for verdadeira,

a *parteTHEN* é executada,

caso contrário,

é executada a *parteELSE*.

```
if (condicao)
        parteTHEN;
   else
        parteELSE;
```

Conta Bancária Segura

Conta Bancária Segura

- Defina em Java uma classe `SafeBankAccount` cujos objectos têm a funcionalidade indicada de seguida.
- Programe a sua classe no BlueJ.
- Teste um (ou vários) objectos `SafeBankAccount`, e verifique se se comportam como esperado.

Conta Bancária Segura

- Uma conta bancária segura é como as contas bancárias que já conhecemos (da classe `BankAccount`).

  Mas a operação de levantamento é segura:

  – O levantamento só é efectuado caso o valor a levantar seja inferior ou igual ao valor do saldo

  – Qualquer tentativa de levantamento de um valor sem provisão na conta leva à aplicação de uma multa

- Uma conta segura fornece ainda outras operações:

  – Para calcular o valor de juros a aplicar

  – Para creditar juros no saldo

- Vamos convencionar também que os valores são registados e armazenados em **cêntimos**

Conta Bancária Segura

- Operações (interface de SafeBankAccount):

```
public void deposit(int amount)
```

Depositar a importância `amount` na conta

```
public void withdraw(int amount)
```

Levantar a importância `amount` na conta ou aplicar a multa

```
public int getBalance()
```

Consultar o saldo da conta

```
public boolean redZone()
```

Indica se a conta está devedora

```
public int computeInterest()
```

Calcular qual o valor do juro anual a aplicar

```
public void applyInterest()
```

Creditar o juro anual ao saldo da conta

Conta Bancária Segura

- Operações (interface de SafeBankAccount):

```
public void deposit(int amount)
```

Depositar a importância `amount` na conta

```
public void withdraw(int amount)
```

Levantar a importância `amount` na conta ou aplicar a multa

```
public int getBalance()
```

Consultar o saldo da conta

```
public boolean redZone()
```

Indica se a conta está devedora

```
public int computeInterest()
```

Calcular qual o valor do juro anual a aplicar

```
public void applyInterest()
```

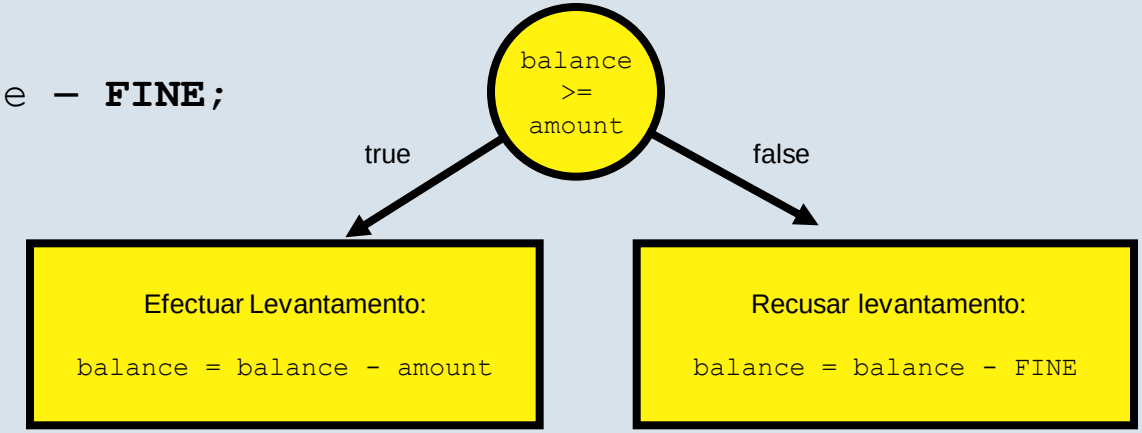
Creditar o juro anual ao saldo da conta

Como programar o método `withdraw` ?

Método `withdraw(int amount)`

- A operação pode ser programada usando a instrução **`if-else`**, do seguinte modo:

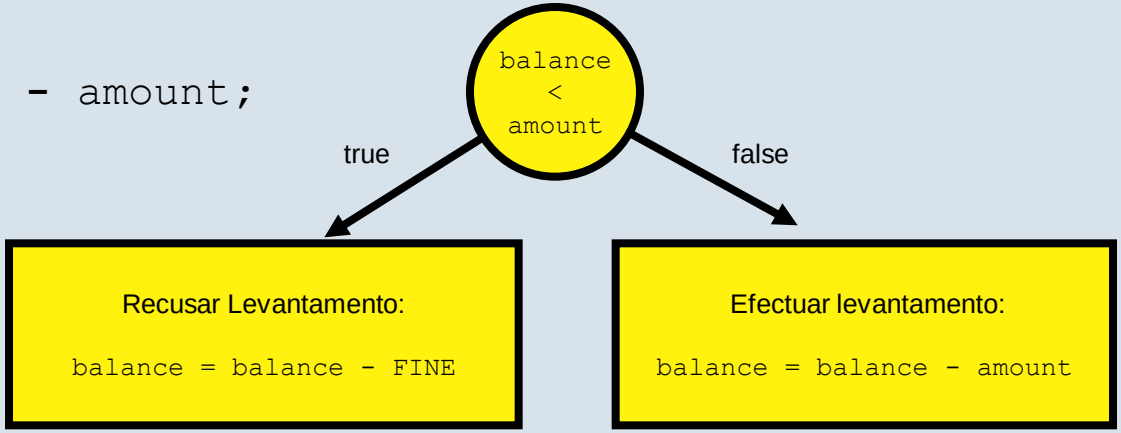
```
public void withdraw(int amount) {
   if (balance >= amount)
      balance = balance - amount;
   else
      balance = balance – FINE;
}
```



Método `withdraw(int amount)`

- Outro modo equivalente de definir a decisão, com base na condição oposta:

```
public void withdraw(int amount) {
   if (balance < amount)
     balance = balance – FINE;
   else
     balance = balance - amount;
}
```



Método `withdraw(int amount)`

- O valor `FINE` da multa é **constante**, não devendo nunca ser alterado durante a execução do programa
- O valor `FINE` é **comum** a todas as contas geradas pela classe `SafeBankAccount`
- Para facilitar eventuais re-programações do valor da multa, é conveniente declarar o mesmo como uma constante global

```
public class SafeBankAccount {
   public static final int FINE = 200;
   ...
}
```

Relembre: Declaração de Constantes

- Constantes Globais:
  - Globais, pois são usadas por **todos** os objectos de uma certa classe
  - Constantes, pois o seu valor nunca deverá ser alterado (é fixo)
- Exemplo: o identificador **`PI`** é uma constante, de tipo **`double`**, definida na classe **`Math`**.
- Para declarar numa classe uma constante, utiliza-se a forma

```
static final tipo constantIdentifier = expr ;
```

A expressão ***expr*** só pode mencionar outras constantes. Por exemplo,

```
static final double PI2 = 2*Math.PI;

static final int FINE = 200;
```

Conta Bancária Segura

Outras operações de **`SafeBankAccount`**:

`public int` `computeInterest()`

Usualmente, o banco credita um certo juro nas contas dos seus clientes, numa base periódica (por exemplo, uma vez por ano)

O valor do juro é calculado por aplicação de uma taxa ao valor do saldo

A taxa de juro é determinada com base no valor do saldo, através de um sistema de escalões (quanto maior o saldo, maior a taxa)

A operação `computeInterest()` determina qual o valor do juro a aplicar tendo em conta o saldo corrente da conta

`public void` `applyInterest()`

Creditar efectivamente o juro apropriado na conta

Determinação da taxa de juro

A taxa de juro a aplicar depende do valor saldo existente na conta.

Existem três taxas possíveis:

Valor do Saldo	Taxa
≤ 2000 €	1%
]2000 €,10.000 €]	2%
> 10.000 €	3%

Método `computeInterest()`

- A operação `computeInterest()` pode ser decomposta em duas tarefas mais elementares
  - Determinar a taxa de juro a partir do saldo corrente
  - Aplicar a taxa ao saldo corrente

```
public int computeInterest() {

    Determinar qual a taxa de juro

    Calcular o valor do juro com
    base no valor do saldo

}
```

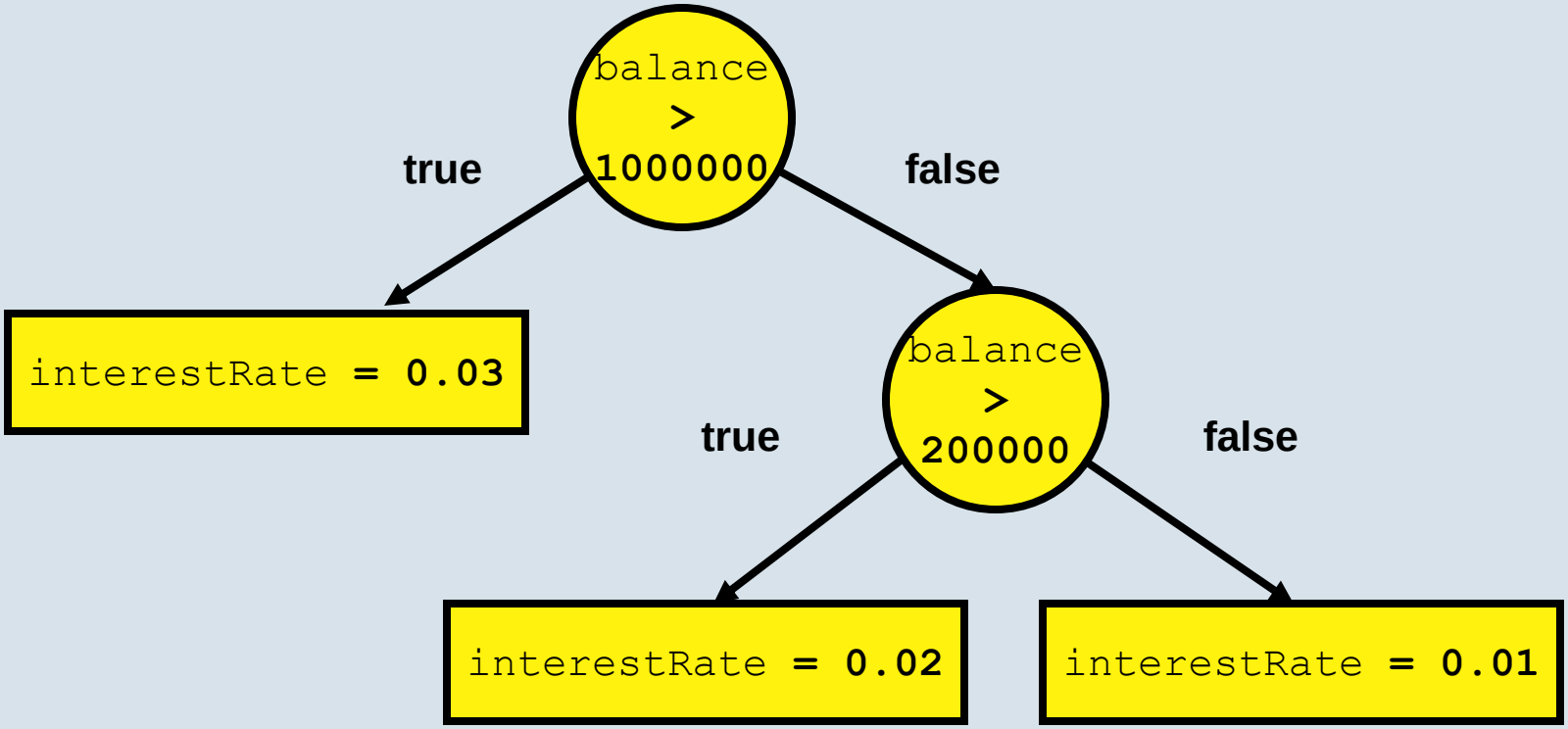
Método `computeInterest()`

- A operação `computeInterest()` pode ser decomposta em duas tarefas mais elementares
  - Determinar a taxa de juro a partir do saldo corrente
  - Aplicar a taxa ao saldo corrente

```
public int computeInterest() {

    interestRate = ?

    Calcular o valor do juro com
        base no valor saldo

}
```

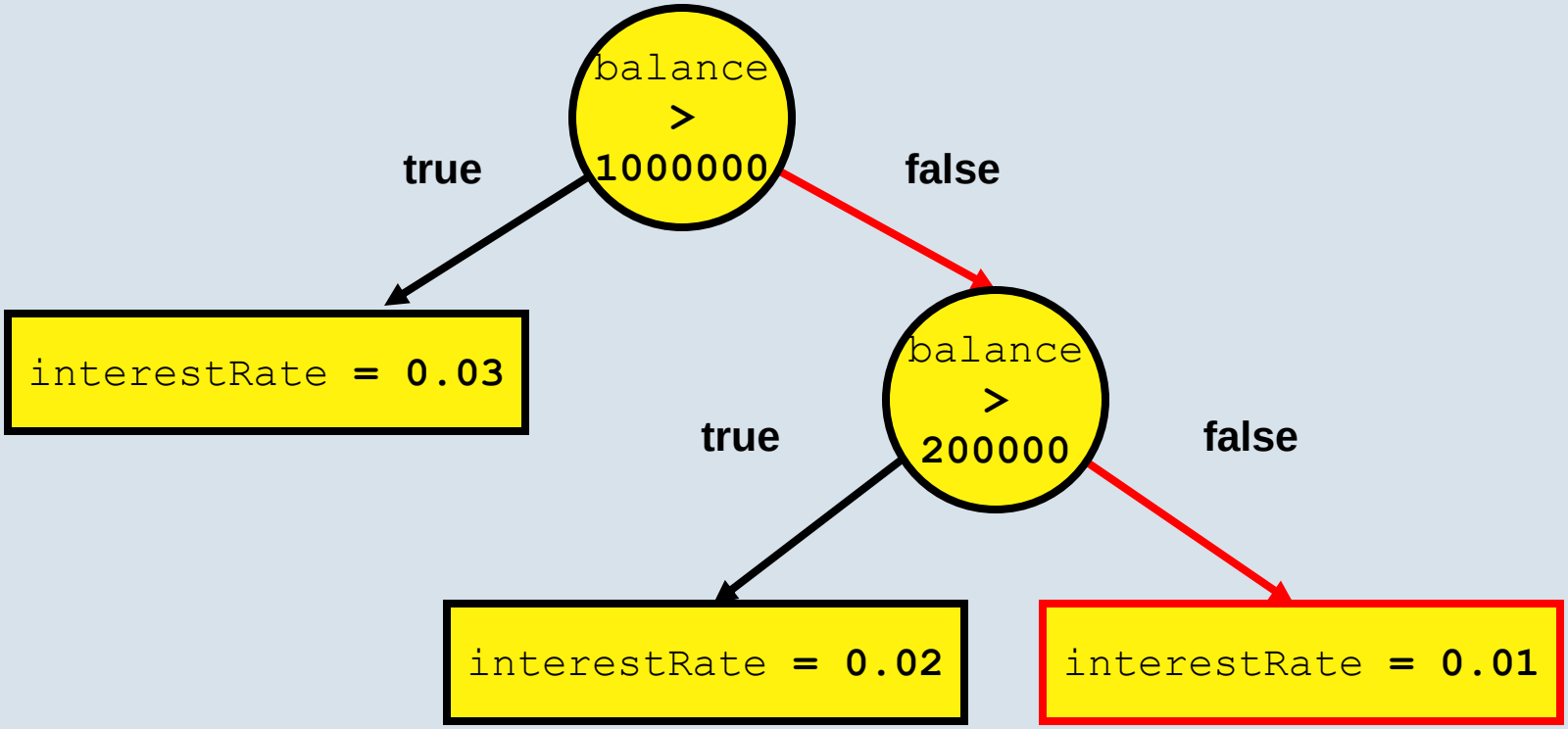
Determinação da taxa de juro

- A determinação da taxa de juro pode ser efectuada através das decisões seguintes (valor em cêntimos):



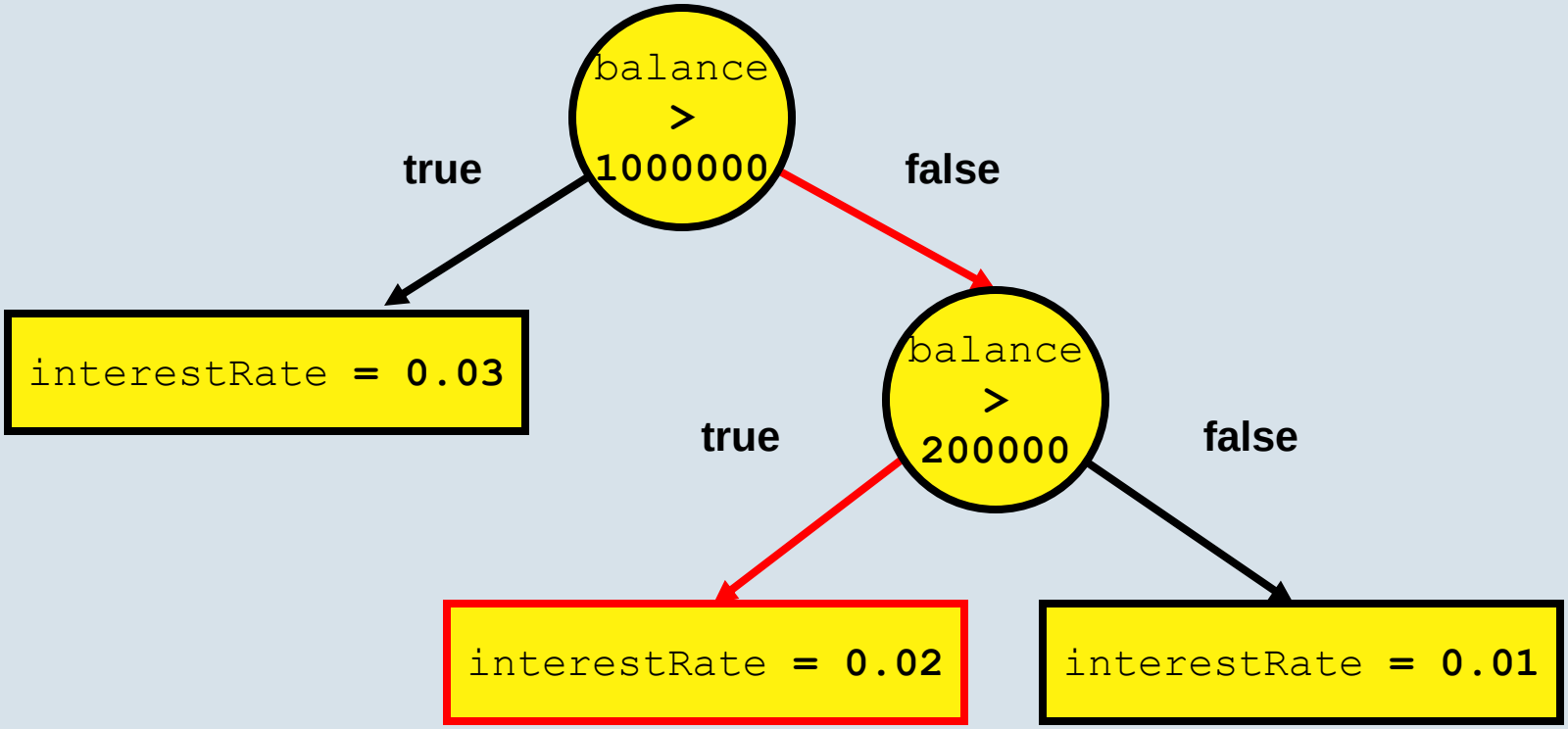
Determinação da taxa de juro

- Exemplo (em cêntimos):
  - `balance = 6700`



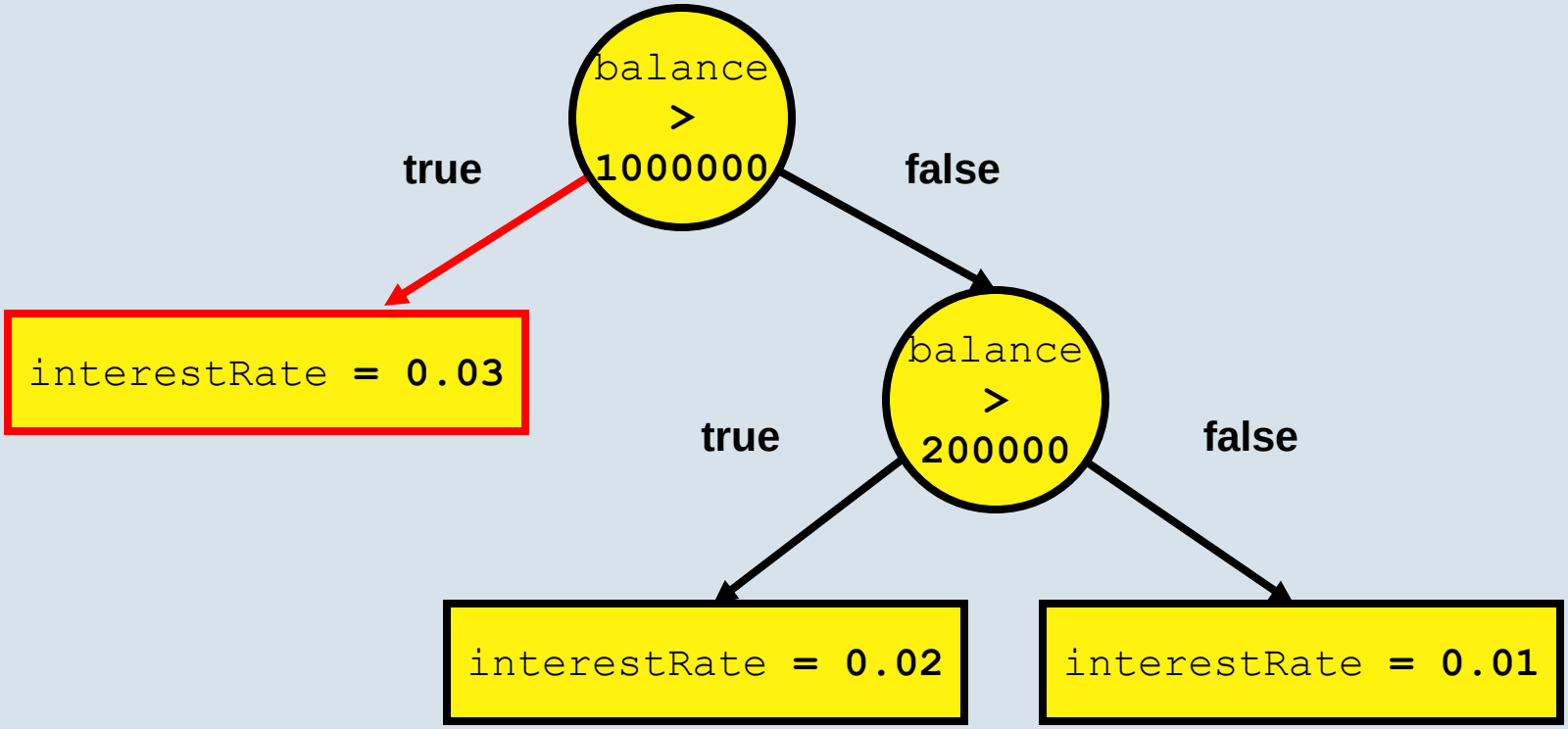
Determinação da taxa de juro

- Exemplo (em cêntimos):
  - `balance = 250000`



Determinação da taxa de juro

- Exemplo (em cêntimos):
  - `balance = 15000000  /* 150000 euros */`



Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
    float interestRate ;

    interestRate = ?

    Calcular o valor do juro com
    base no saldo
}
```

Declaração de variável local

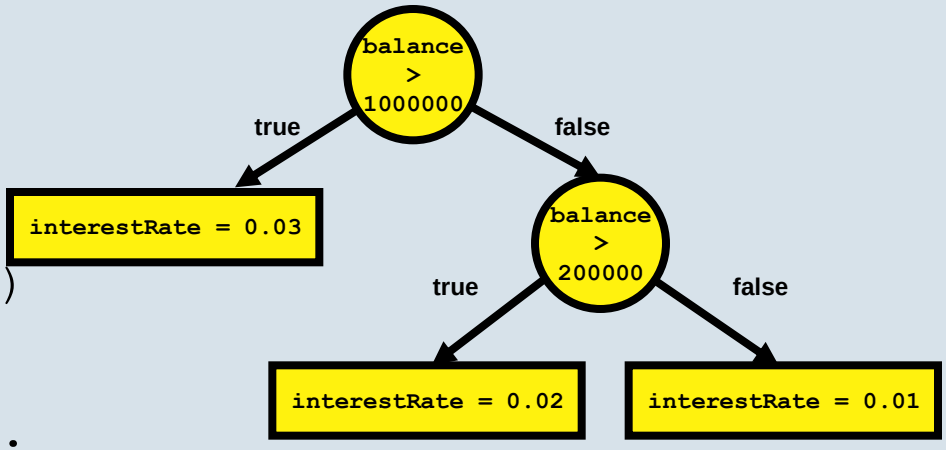
O valor de `interestRate` tem que ser determinado a partir do valor do saldo

Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
   float interestRate ;
   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;
}
```

Calcular o valor do juro com base no saldo

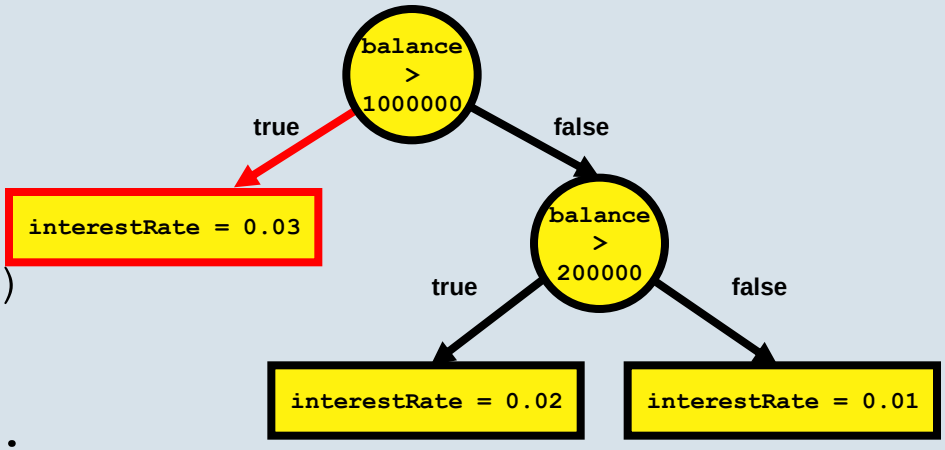


Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
   float interestRate ;
   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;
}
```

Calcular o valor do juro com base no saldo

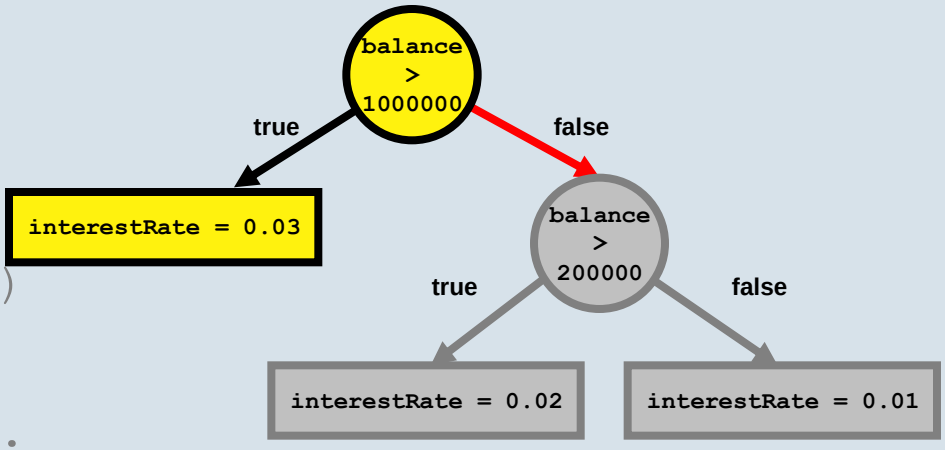


Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
   float interestRate ;
   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;
}
```

Calcular o valor do juro com base no saldo

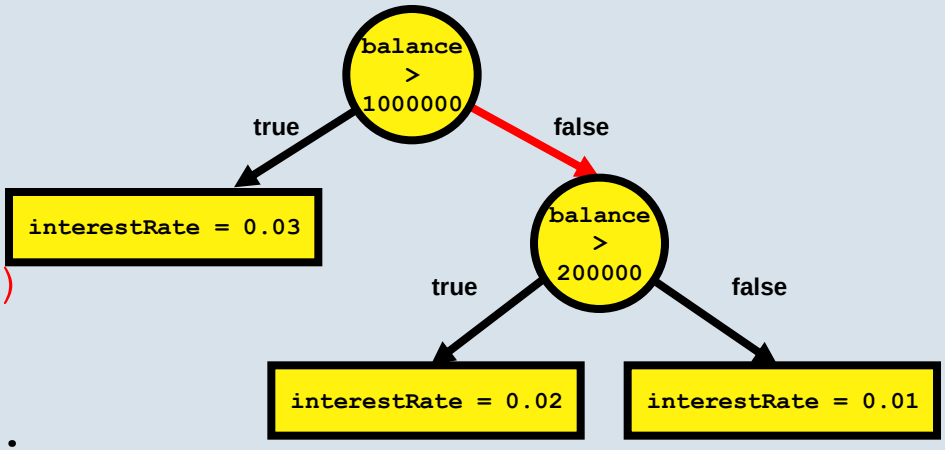


Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
   float interestRate ;
   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;
}
```

Calcular o valor do juro com base no saldo

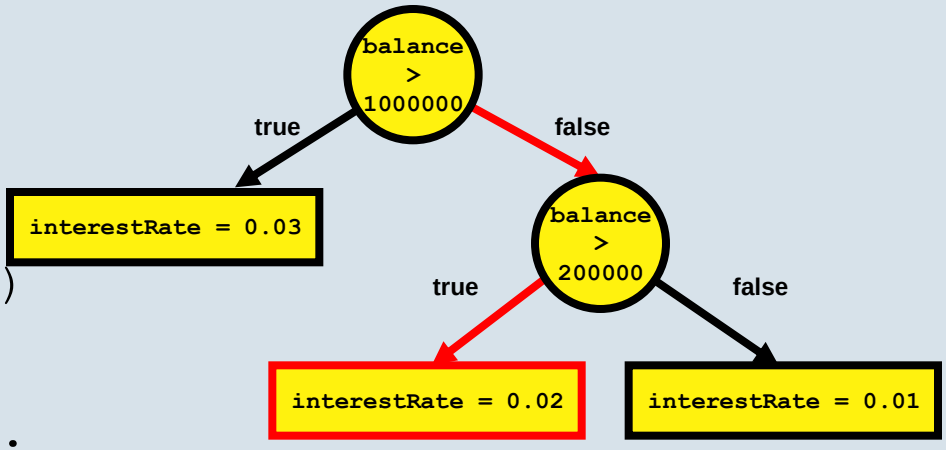


Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
   float interestRate ;
   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;
}
```

Calcular o valor do juro com base no saldo

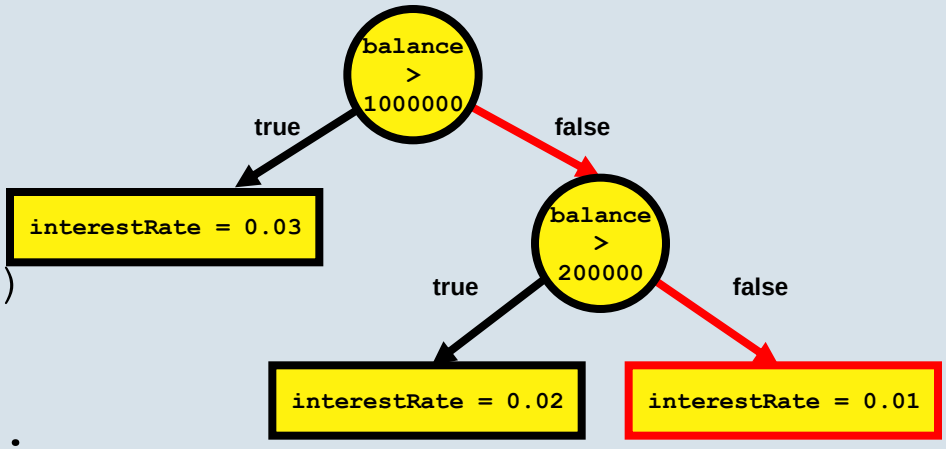


Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
   float interestRate ;
   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;
}
```

Calcular o valor do juro com base no saldo



Método `computeInterest()`

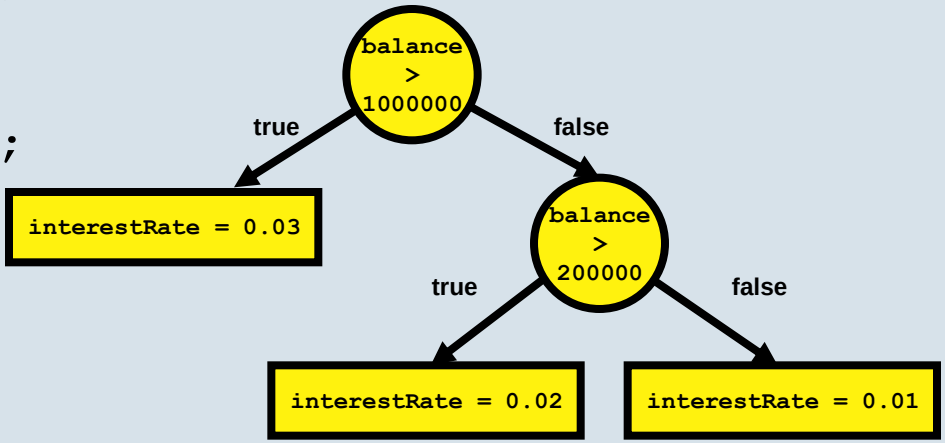
- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
   float interestRate ;
   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;
```

Calcular o valor do juro com base no saldo

```
}
```

Valor do Saldo	Taxa
≤ 2000 €	1%
]2000 €,10.000 €]	2%
> 10.000 €	3%



Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
   float interestRate ;

   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;

   return balance * interestRate;
}
```

Determinar qual a taxa a partir do valor do saldo

Calcular o valor do juro com base no saldo e na taxa

Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
   float interestRate ;

   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;

   return balance * interestRate;
}
```

Determinar qual a taxa a partir do valor do saldo

Esta expressão vai produzir um valor de tipo **`float`** (**`float * int ⇒ float`**)

Calcular o valor do juro com base no saldo e na taxa

Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
   float interestRate ;

   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;

   return Math.round(balance * interestRate);
}
```

Determinar qual a taxa a partir do valor do saldo

Como estamos a representar os valores em cêntimos (tipo **`int`**) é necessário arredondar e converter o resultado para o tipo **`int`**

Método `computeInterest()`

- Método `computeInterest()`

```
public int computeInterest() {
   float interestRate ;
   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;
   return Math.round(balance * interestRate);
}
```

- Esta solução funciona, mas devemos fazer melhor!

Método `computeInterest()`

- Método `computeInterest()`

```
public int computeInterest() {
   float interestRate ;
   if (balance > 1000000)
    interestRate = 0.03f;
   else if (balance > 200000)
    interestRate = 0.02f;
   else interestRate = 0.01f;
   return Math.round(balance * interestRate);
}
```

Deve-se evitar utilizar constantes "mágicas" nos programas, pois tal torna os programas difíceis de ler e de modificar.

- Técnica correcta: dar nomes às constantes na classe.

Método `computeInterest()`

- Método `computeInterest()`

```
public int computeInterest() {
   float interestRate ;
   if (balance > CAT1)
    interestRate = RATECAT1;
   else if (balance > CAT2)
    interestRate = RATECAT2;
   else interestRate = RATECAT3;
   return Math.round(balance * interestRate);
}
```

Deve-se evitar utilizar constantes "mágicas" nos programas, pois tal torna os programas difíceis de ler e de modificar.

- Técnica correcta: dar nomes às constantes na classe.

Método `computeInterest()`

- Técnica correcta: dar nomes às constantes na classe

```
public class SafeBankAccount {
   ...
   public static final int CAT1 = 1000000;
   public static final int CAT2 = 200000;
   public static final float RATECAT1 = 0.03f;
   public static final float RATECAT2 = 0.02f;
   public static final float RATECAT3 = 0.01f;
   ...
}
```

Tipo **`boolean`** e Operações Associadas

- Operações que produzem um valor booleano (**`true`** ou **`false`**) a partir de valores escalares (de tipo **`int`**, **`float`** or **`double`**)
- As expressões *expr1* e *expr2* têm que ser **`int`**, **`float`** or **`double`**

expr1 `>` *expr2*	maior que
expr1 `>=` *expr2*	maior ou igual
expr1 `<` *expr2*	menor
expr1 `<=` *expr2*	menor ou igual
expr1 `==` *expr2*	igualdade
expr1 `!=` *expr2*	desigualdade

- **Importante**: Estes operadores chamam-se **operadores relacionais**.

Tipo **`boolean`** e Operações Associadas

- Operações que produzem um valor booleano (de tipo **`boolean`**) a partir de valores booleanos (de tipo **`boolean`**)
- Aqui, vamos assumir que *expr1* e *expr2* têm que ser **`boolean`**

expr1 **`&&`** *expr2*	conjunção lógica
expr1 **`||`** *expr2*	disjunção lógica
`!` *expr*	negação lógica
`true`	valor lógico “verdade”
`false`	valor lógico “falso”

- **Importante**: Estes operadores chamam-se **operadores lógicos**.

Conta Bancária Segura

Outras operações de **`SafeBankAccount`**:

`public int` `computeInterest()`

Usualmente, o banco credita um certo juro nas contas dos seus clientes, numa base periódica (por exemplo, uma vez por ano)

O valor do juro é calculado por aplicação de uma taxa ao valor do saldo

A taxa de juro é determinada com base no valor do saldo, através de um sistema de escalões (quanto maior o saldo, maior a taxa)

A operação `computeInterest()` determina qual o valor do juro a aplicar tendo em conta o saldo corrente da conta

`public void` `applyInterest()`

Creditar efectivamente o juro apropriado na conta

Método `applyInterest()`

- O método `applyInterest()` pode chamar o método `computeInterest()` e acumular o resultado no saldo

```
public void applyInterest() {
  balance = balance + this.computeInterest();
}
```

- Em geral, para chamar um método do próprio objecto no corpo de um método aplicamo-lo ao nome especial **`this`**
  - O identificador **`this`** refere o **próprio objecto**, neste caso, o mesmo objecto em que a operação `applyInterest()` está a ser executada

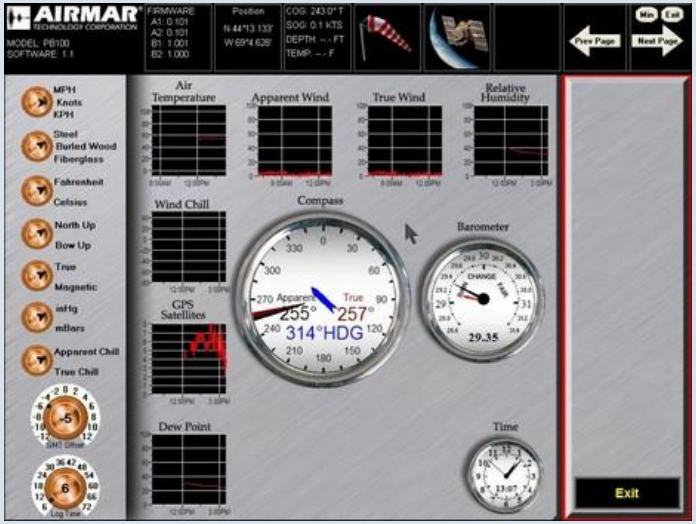
Classe SafeBankAccount

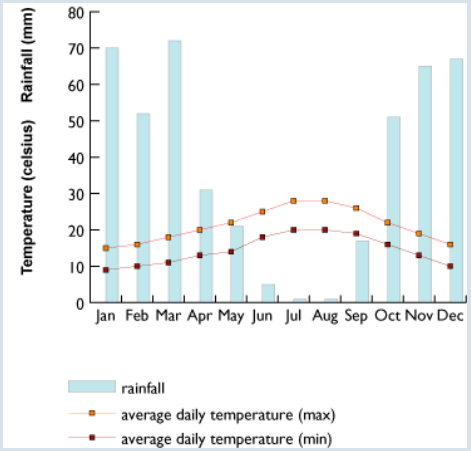
```
SafeBankAccount ba1 = new SafeBankAccount(1000);
ba1.getBalance()
1000   (int)
ba1.withdraw(1500);
ba1.getBalance()
800   (int)
ba1.computeInterest()
8   (int)
ba1.applyInterest();
ba1.getBalance()
808   (int)
```

Estação Meteorológica

Estação Meteorológica

- Defina em Java uma classe `WeatherStation` cujos objectos representam uma estação meteorológica.
- Programe a sua classe no BlueJ.
- Teste um (ou vários) objectos `WeatherStation`, e verifique se se comportam como se esperada.





Estação Meteorológica

- Cada objecto `WeatherStation` recebe valores de temperatura ao longo do tempo e guarda o máximo, o mínimo e a média das temperaturas registadas
- Operações reconhecidas

```
public void sampleTemperature(double temp)
```

Registar a amostra `temp` na estação

```
public double getMaximum()
```

Consultar a máxima temperatura observada até ao momento

```
public double getMinimum()
```

Consultar a mínima temperatura observada até ao momento

```
public double getAverage()
```

Consultar a média das temperaturas observadas até ao momento

```
public void reset(double temp)
```

Apagar a informação registada na estação, para se iniciarem novas medições. É indicada também uma primeira nova medição.

Estação Meteorológica

```
WeatherStation alaska = new WeatherStation(10);
alaska.getMinimum()
10.0    (double)
alaska.sampleTemperature(12):
alaska.sampleTemperature(13);
alaska.sampleTemperature(16);
alaska.sampleTemperature(9);
alaska.getMaximum()
16.0    (double)
alaska.getMinimum()
9.0    (double)
alaska.getAverage()
12.0    (double)
alaska.reset(-1);
alaska.getAverage()
-1.0    (double)
```

Quando a estação é activada, fornece-se uma amostra inicial da temperatura (10 neste caso)

Jogo da Adivinha

Adivinha o Número Secreto

- Defina em Java uma classe `GuessWhat` cujos objectos são jogos de adivinhar números.
- Programe a sua classe no BlueJ.
- Teste um (ou vários) objectos `GuessWhat`, e verifique se se comportam como se espera.

Jogo da Adivinha

- Ouando um jogo é criado, indica-se o valor mínimo e máximo do número secreto (a adivinhar)
- Operações reconhecidas

```
public String tryit(int guess)
```

O jogador usa este método para apostar que o número secreto é `guess`.

O método `tryit` responde (devolve)

"You win!"	se o número proposto é o certo
"Too high!"	se o número proposto é maior que o secreto
"Too low!"	se o número proposto é menor que o secreto
"Game Over"	se o número proposto não faz sentido ... Pois já foi excluído por jogadas anteriores

```
public void newGame()
```

Iniciar novo jogo

Jogo da Adivinha

```
GuessWhat game = new GuessWhat(0,9);
game.tryit(5)
"Too low!"    (String)
game.tryit(8)
"Too high!"   (String)
game.tryit(9)
"Game Over"   (String)
game.newGame();
game.tryit(5)
"Too high!"   (String)
game.tryit(3)
"Too low!"    (String)
game.tryit(4)
"You win!"    (String)
```

Para programar a classe `GuessWhat` vai necessitar da função

`Math.rand()`

A função `Math.rand()` devolve um número aleatório **`double`** no intervalo [0,1[